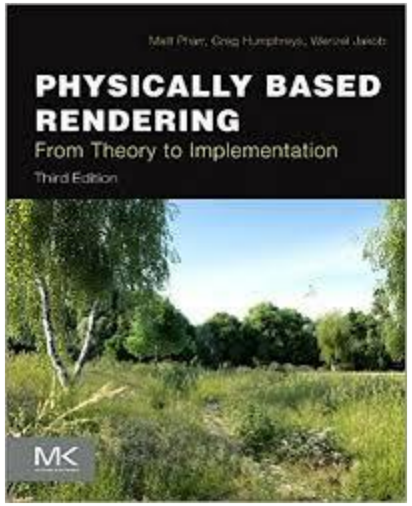
Introduction to Ray Tracing

CS 296: Data Structures Honors Section
Geometric Data Structures for Computer Graphics

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Eric Shaffer

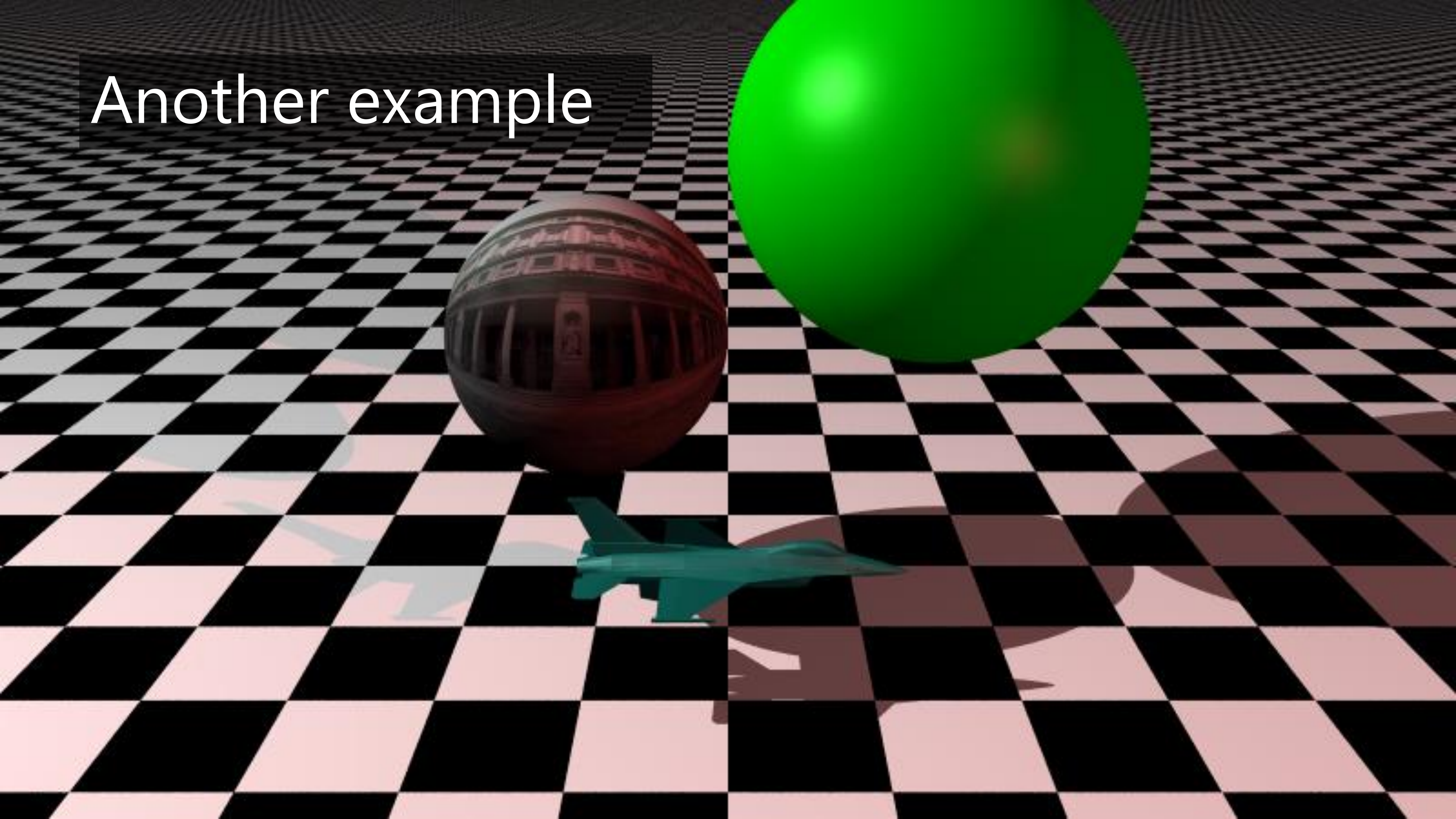
Great Moments in Computer Graphics…



- [Academy Award for Physcially-Based Rendering]

Rendering by Simulating Light Transport

Another example



Some Research Areas in Rendering

- Increase quality
  - More accurate modeling of the physics of light
  - Better models of materials

- Reduce time
  - More efficient algorithms (numerical or otherwise)
  - **Better data structures**
  - Parallelism

Managing Geometric Complexity

PantaRay: Fast Ray-Traced Occlusion Caching of Massive Scenes Pantaleoni et al., SIGGRAPH 2010

Some Stuff about the Class

Syllabus is on course website:

https://courses.engr.illinois.edu/cs225/sp2018/honors/

- We'll use Piazza… https://piazza.com/class/jc81qjz1pa161b
  - use the *honors* folder

Some Stuff about the Class

- Grades probably on usual scale:
  - 97 to 93: A
  - 93 to 90: A-
  - 90 to 87: B+
  - 87 to 83: B
  - 83 to 80: B-
  - …etc.
- won’t be any tighter

Some Stuff about the Class

All programming assignments
…you should have the chance to build something

- MP1: 40%
- MP2: 30%
- MP3: 30%
- Extra Credit MP

Stuff about the Class

Recommended (Required?) texts:

$2.99 each on Amazon (Kindle edition)

Peter Shirley

From Wikipedia, the free encyclopedia

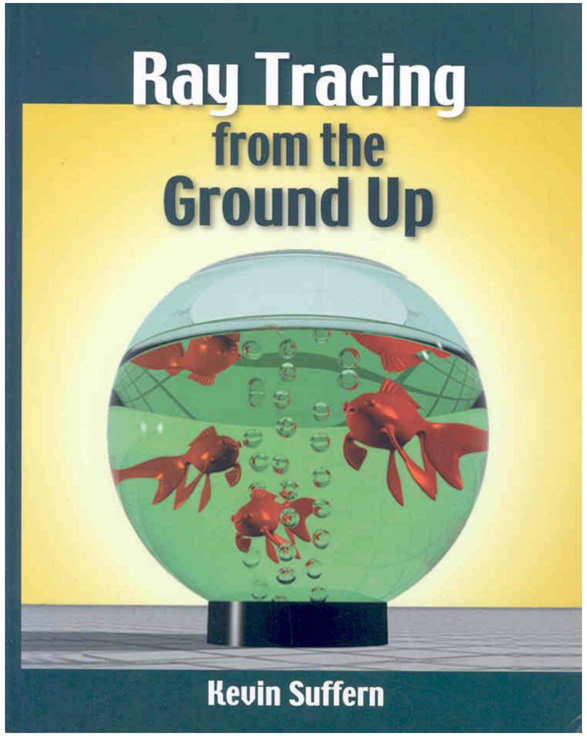
Peter Shirley (born 1963) is American computer scientist and computer graphics researcher. He is a Distinguished Scientist at NVIDIA and adjunct professor at the University of Utah in computer science. He has made extensive contributions to interactive photorealistic rendering.[1] His textbook, *Fundamentals of Computer Graphics*, is considered one of the leading introductory texts on computer graphics and is currently in the fourth edition.[2][3]

Biography [edit]

Shirley received his BA in physics from Reed College in 1985, and his PhD in computer science from the University of Illinois, Urbana-Champaign in 1991.[4] He then joined the faculty at Indiana University as an assistant professor. From 1994 to 1996 he was a visiting professor at Cornell University. He then joined the University of Utah, where he taught until 2008 when he joined NVIDIA as a research scientist.

Some Stuff about the Class

Recommended text:



Some Stuff about Coding

- Use C++
- Use the libpng library to save images (if you want)
- If you want, use libglm for math….
- Help each other out on Piazza

- You can use other people's code.
  - You have to document your use
  - Failure to do so results in a 0 on the assignment
- Plagiarism policy: **Type it in yourself**

Our Model of Light Rays

Three key ideas about light rays

- Light travels in straight lines
- Light rays do not interfere with each other if they cross
- Light rays travel from light sources to the eye,
  but the physics is invariant under path reversal–reciprocity

Ray Tracing

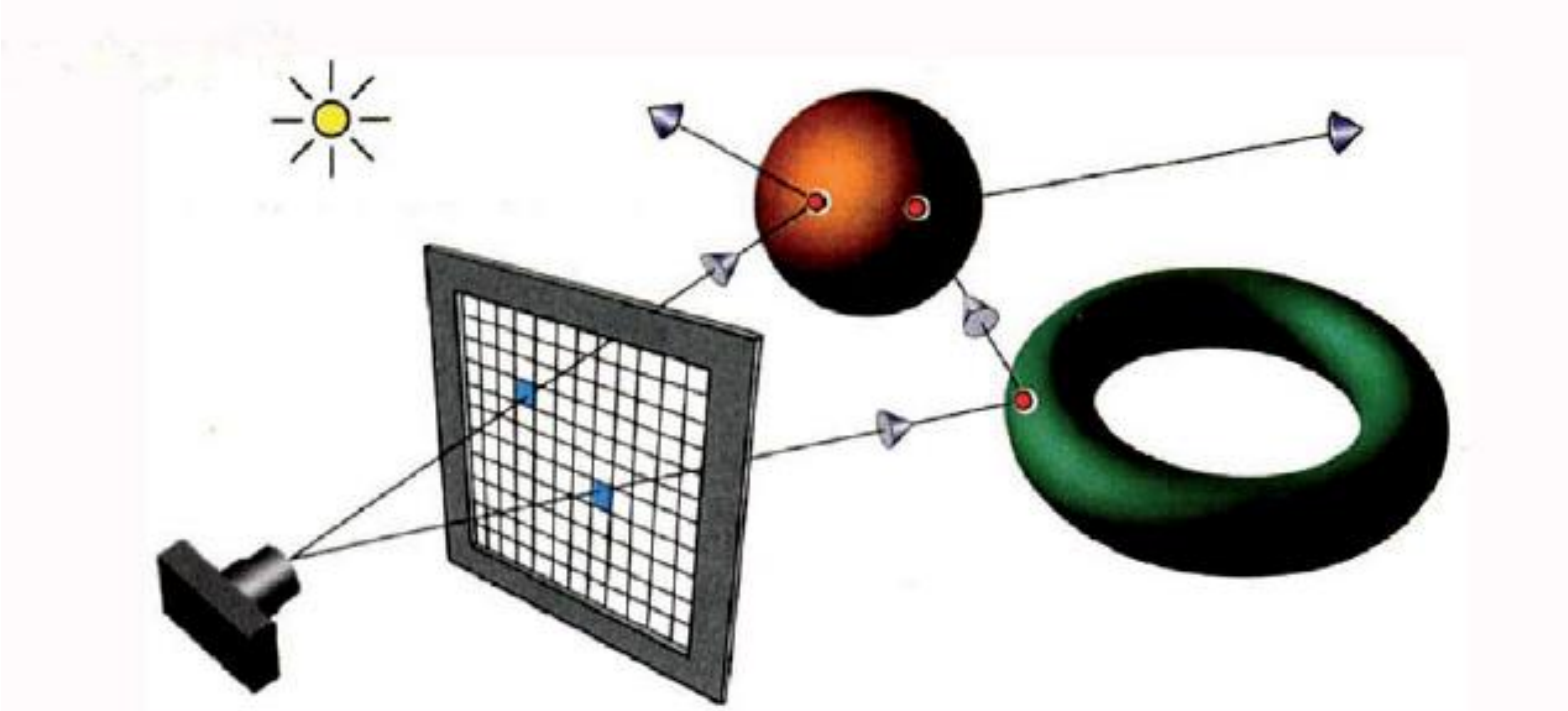
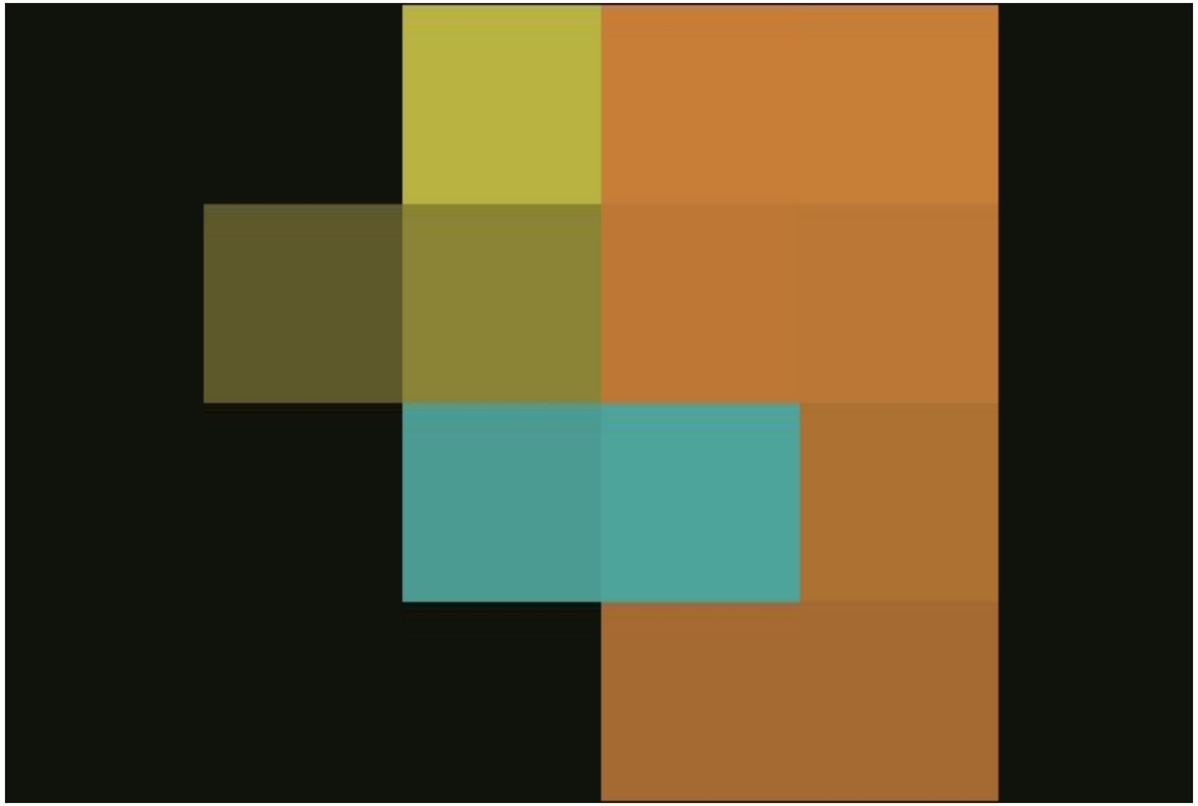
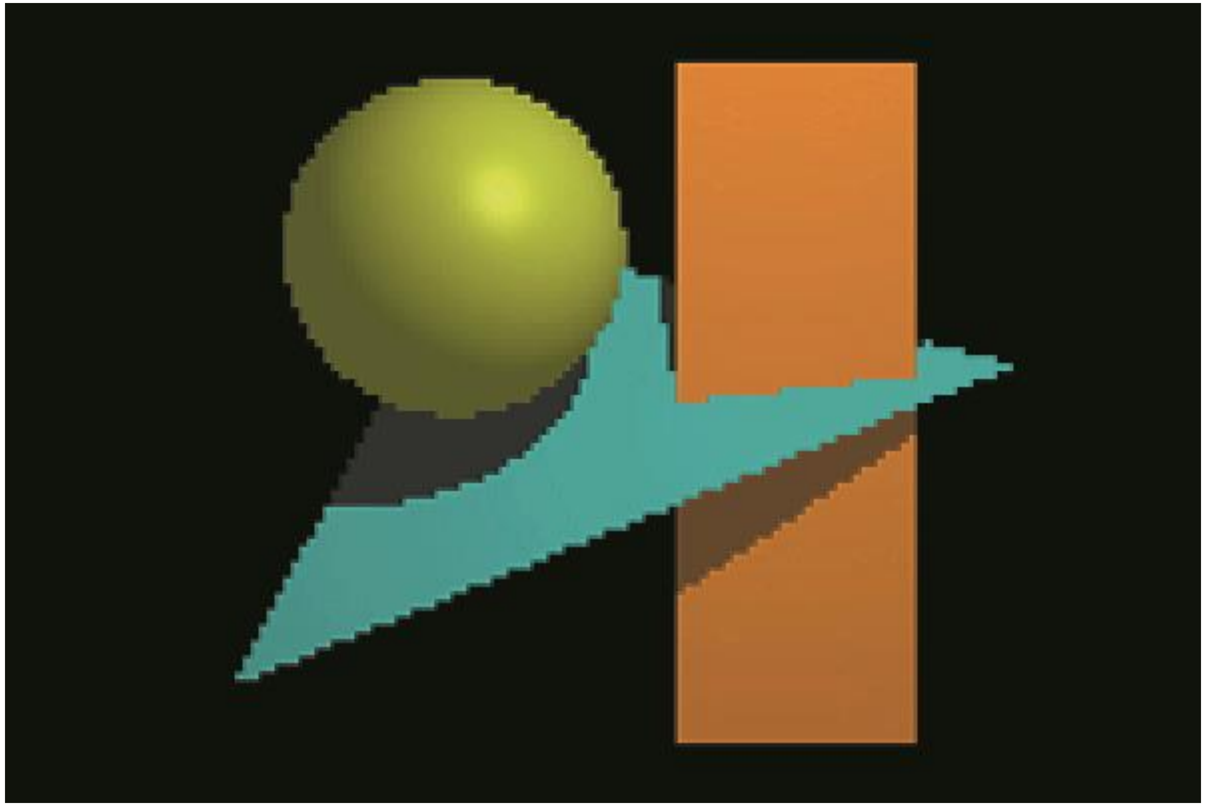


Figure from *Ray Tracing from the Ground Up* by Kevin Suffern

Resolution





Ray Tracing – basic algorithm

1. define some objects
2. specify a material for each object
3. define some light sources
4. define window that consists of a grid of pixels (the view plane)

5. for each pixel
6. shoot a ray into the model world

7. compute the intersection of the ray with each object

8. find the intersection (if any) closest to the view plane

9. if there was an intersection
10. use lights and material to compute the pixel color
11. else
12. pixel is set to background color

Well…actually we're ray-casting

- We can also cast some other rays to achieve other effects
  - That's when we are able to say we ray-tracing
- Types of rays:
  - Primary rays – rays shot from pixel (or eyepoint) into the world
  - Secondary rays
  - Shadow rays
  - Light rays

Orthographic ray-tracing

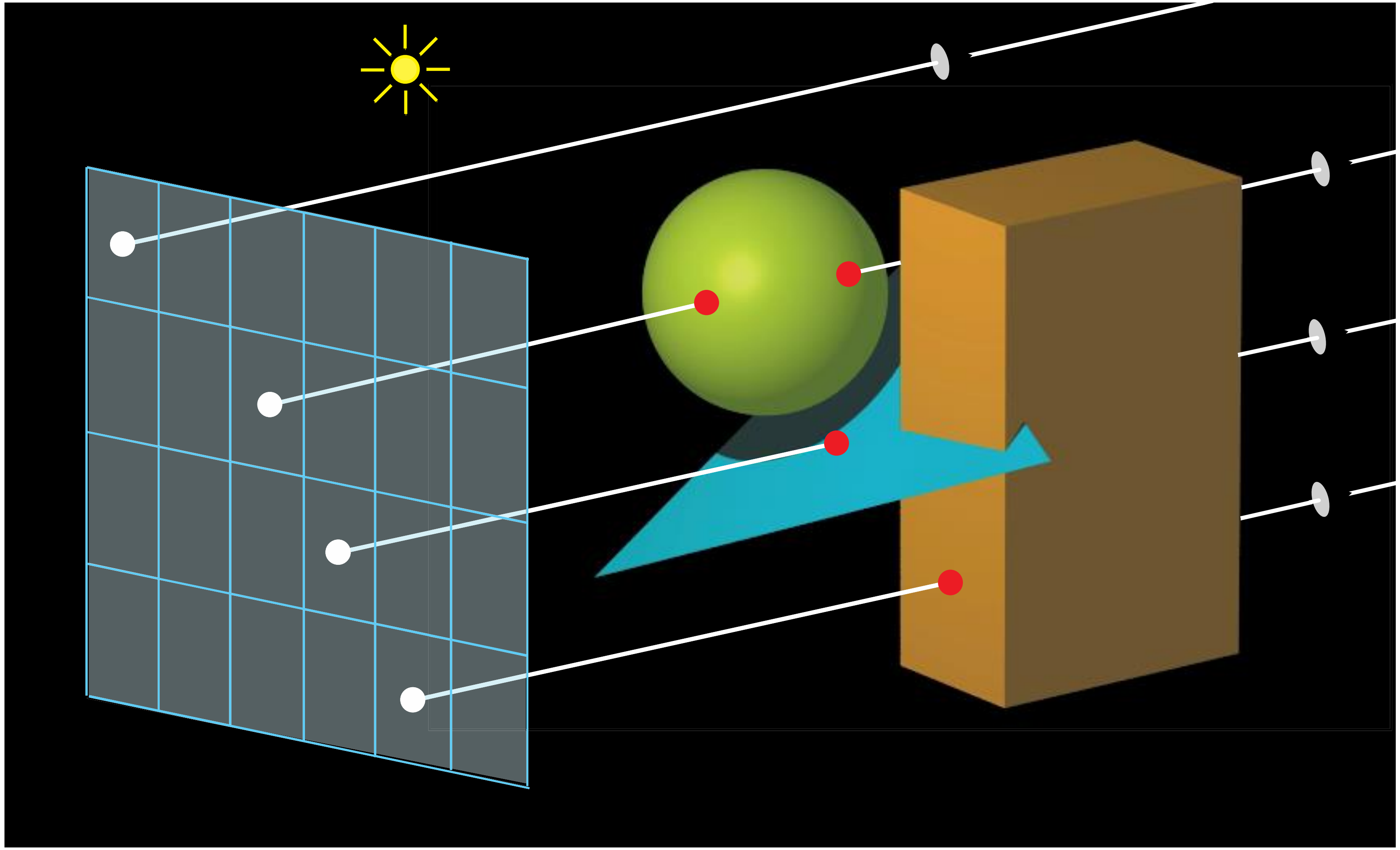


Figure from *Ray Tracing from the Ground Up* by Kevin Suffern

Orthographic ray-tracing

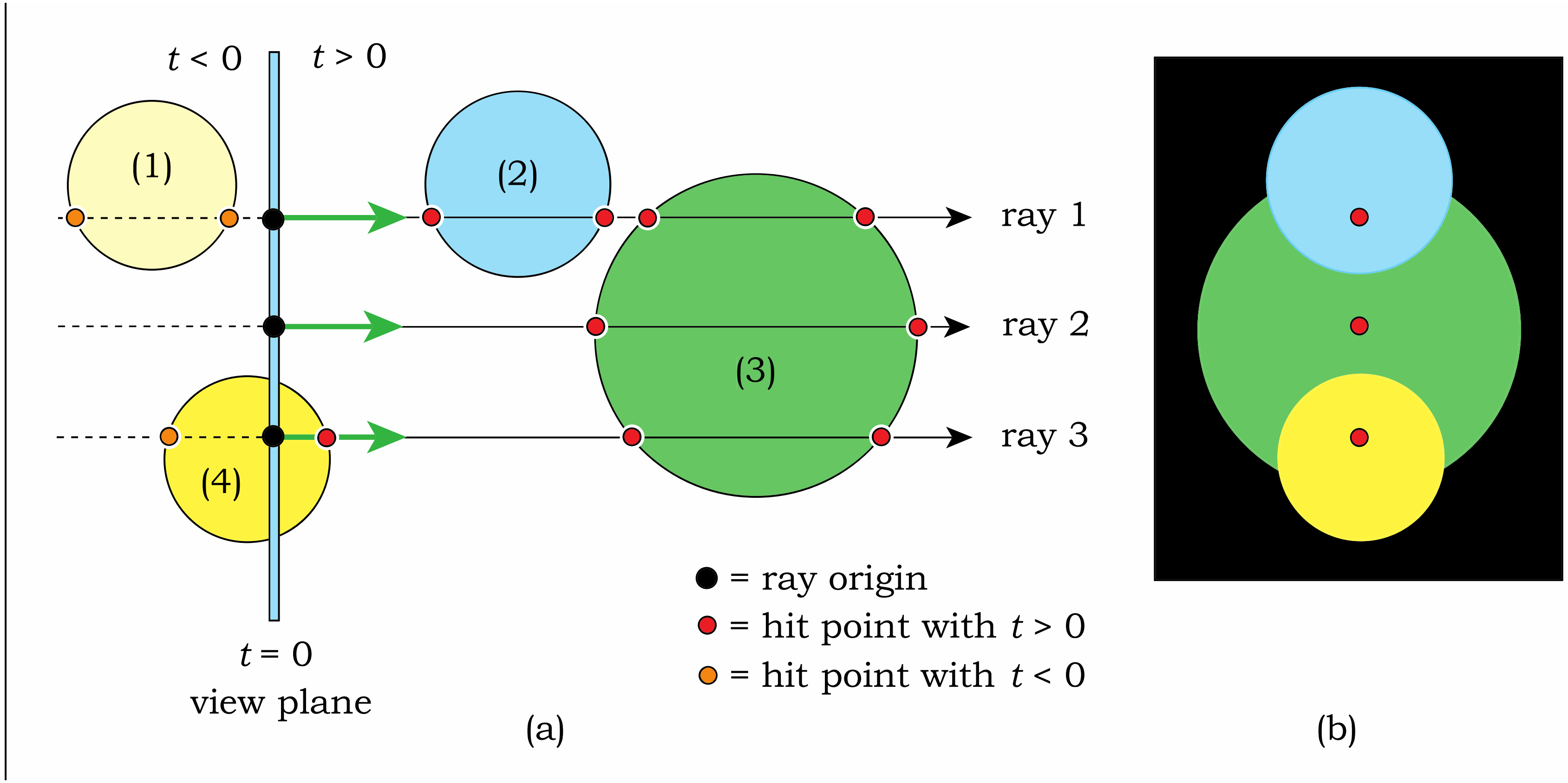
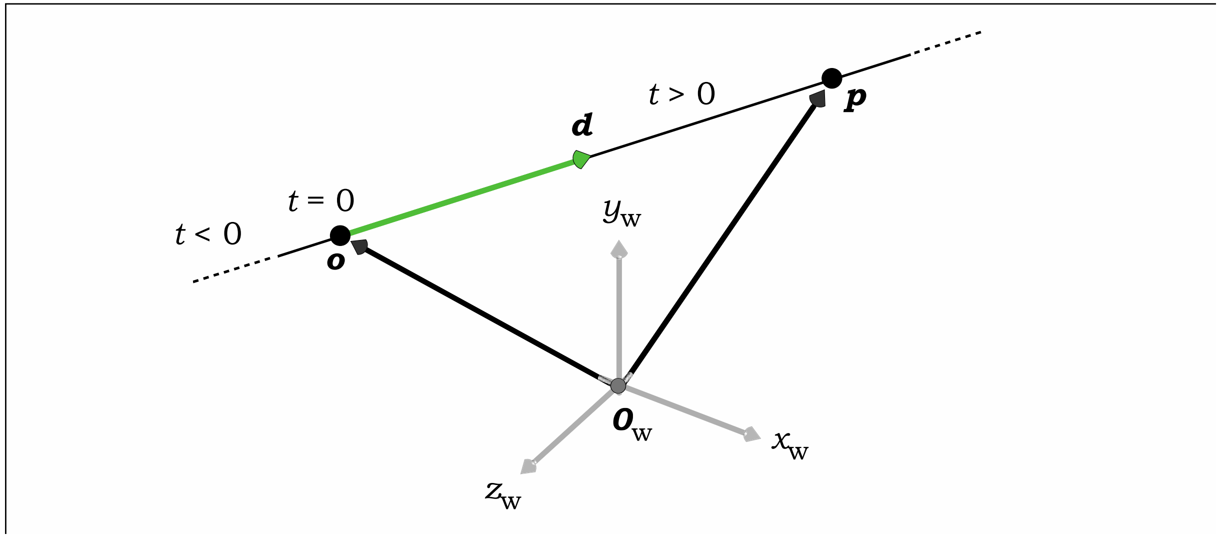


Figure from *Ray Tracing from the Ground Up* by Kevin Suffern

Rays

$$p = o + td$$



p is a point on the ray
o is point that is origin of the ray
t is scalar parameter
d is unit vector giving the direction of the ray

Intersecting Rays and Implicit Surfaces

We can intersect an implicit surface with a ray

Just find the point p on the ray that satisfies the equation for the implicit surface

$$f(x, y, z) = 0$$

$$f(p) = 0$$

$$f(o + td) = 0$$

For now, we'll just focus on planes and spheres….

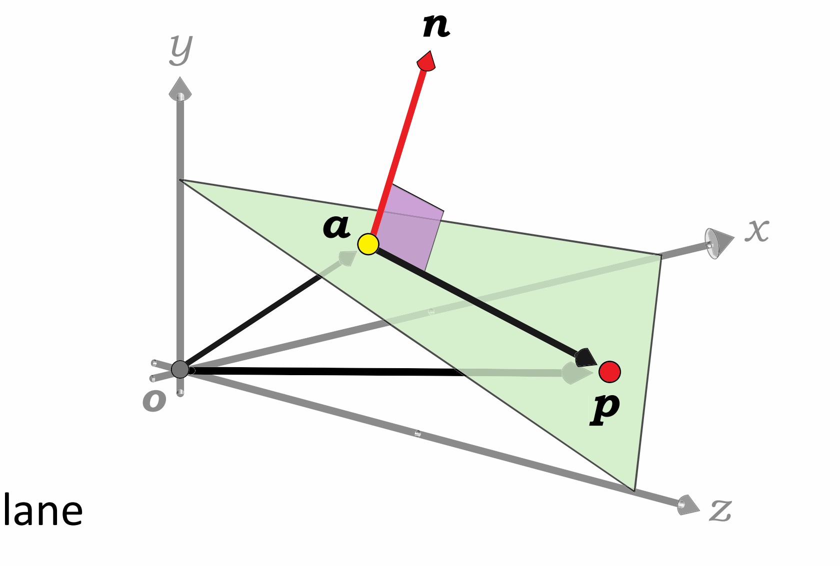
Plane Equation

$$Ax + By + Cz + D = 0$$

$$(p - a) \cdot n = 0$$

Here *a* is a point on the plane
And *n* is the normal

All points *p* that satisfy the equation form the plane



Ray-Plane Intersection

$$(p - a) \cdot n = 0$$

the plane equation with normal n and point on plane a

$$(o + td - a) \cdot n = 0$$

$$t = ((a - o) \cdot n) / (d \cdot n)$$

What happens if d is parallel to the plane?

How do you know if the hit happens in front or behind the view plane?

Ray-Sphere Intersection

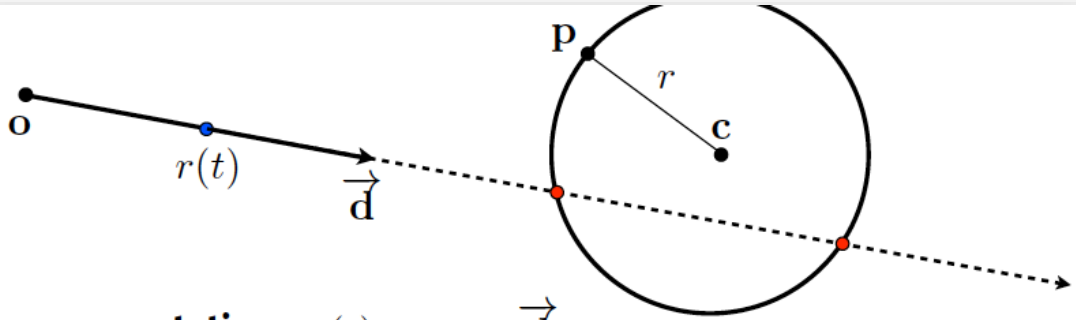
$$(p - c)\cdot(p - c) - r^2 = 0$$ Equation of a sphere

$$(o + td - c)\cdot(o + td - c) - r^2 = 0$$

$$(d \cdot d)t^2 + [2(o - c)\cdot d]t + (o - c)\cdot(o - c) - r^2 = 0$$

So…a quadratic equation that we can solve

Ray-Sphere Intersection



Ray representation: $r(t) = \mathbf{o} + t\,\vec{\mathbf{d}}$

Sphere representation: $||\mathbf{p} - \mathbf{c}||^2 - r^2 = 0$

$$(\mathbf{o} + t\,\vec{\mathbf{d}} - \mathbf{c})^2 - r^2 = 0$$

$$at^2 + bt + c = 0$$

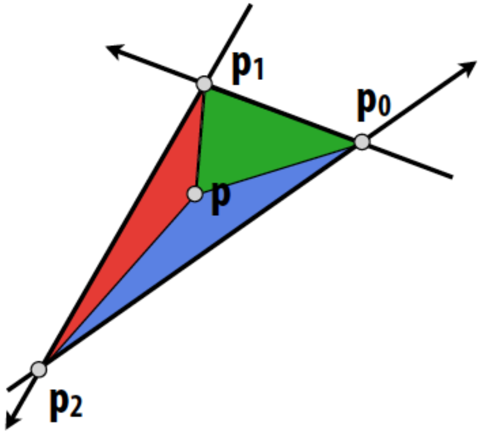
$$a = \vec{\mathbf{d}} \cdot \vec{\mathbf{d}}$$
$$b = 2(\mathbf{o} - \mathbf{c}) \cdot \vec{\mathbf{d}}$$
$$c = ((\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c})) - r^2$$

Slide from *CS348b: Image Synthesis* by Matt Pharr

Point in Triangle Test

Barycentric Coordinates

- $a_0(p) = \text{Area}(p_1, p_2, p)$
  $a_1(p) = \text{Area}(p_2, p_0, p)$
  $a_2(p) = \text{Area}(p_0, p_1, p)$
- Define barycentric coordinates:
  - $b_0 = a_0 / \text{Area}(p_0, p_1, p_2)$
  - $b_1 = a_1 / \text{Area}(p_0, p_1, p_2)$
  - $b_1 = a_2 / \text{Area}(p_0, p_1, p_2)$ ←should be b2
- p is inside the triangle if:
  - $0 \leq b_0 \leq 1$
  - $0 \leq b_1 \leq 1$
  - $0 \leq b_2 \leq 1$



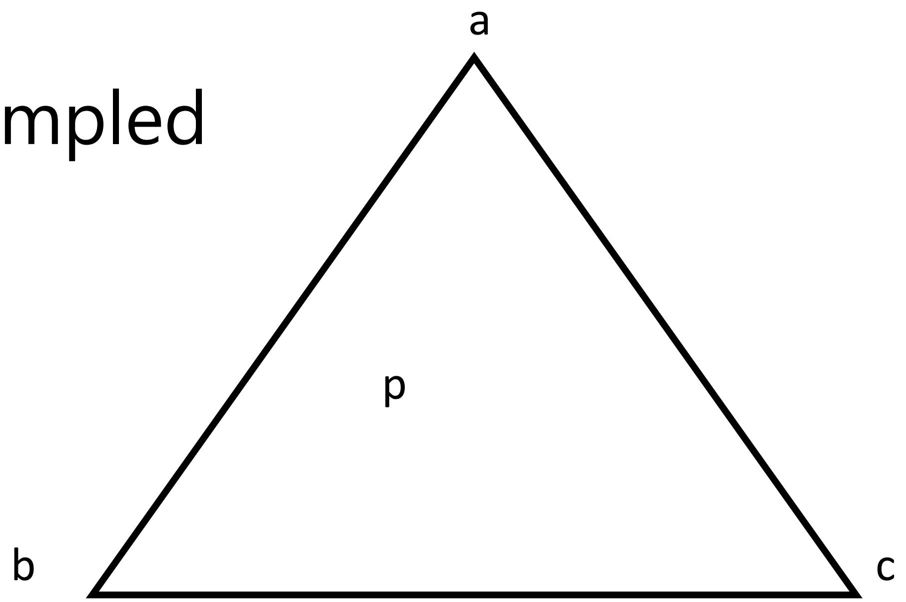
Slide from *CS348b: Image Synthesis* by Matt Pharr

Barycentric Coordinates for Triangles

- Describe location of point in a triangle in relation to the vertices
- $p=(\lambda_1, \lambda_2, \lambda_3)$ where the following are true
  - $p=\lambda_1 a + \lambda_2 b + \lambda_3 c$
  - $\lambda_1 + \lambda_2 + \lambda_3 = 1$
- To interpolate a function sampled

at the vertices we just do:

$\mathbf{f(p)=\lambda_1 f(a) + \lambda_2 f(b) + \lambda_3 f(c)}$



Barycentric Coordinates for Triangles

The area of triangle abc can be found by cross product:

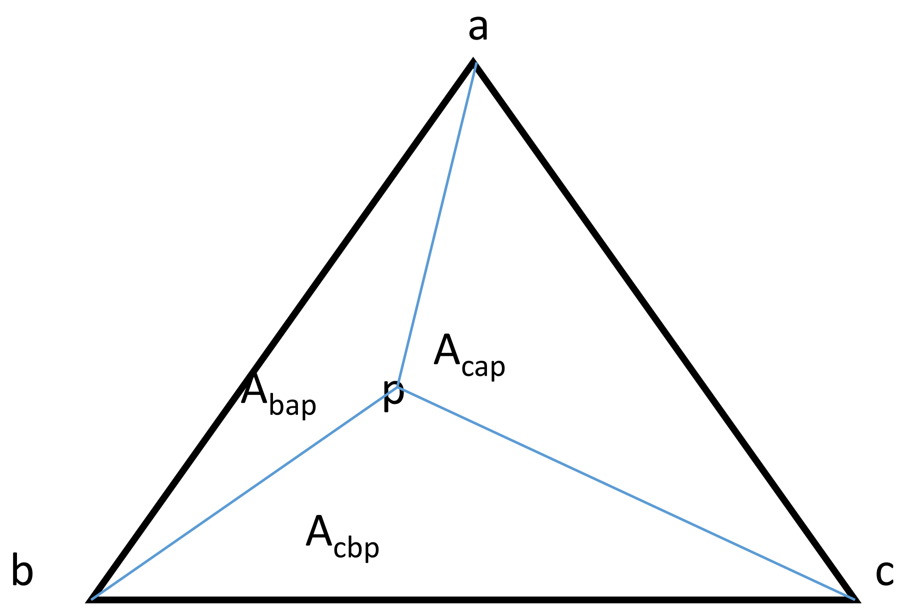
$$AREA_{abc} = \|(b-a) \times (c-a)\|/2$$

So we have:

$\lambda_1 = AREA_{cbp}/AREA_{abc}$
$\lambda_2 = AREA_{cap}/AREA_{abc}$
$\lambda_3 = AREA_{bap}/AREA_{abc}$

How can we optimize computation of λ_3?

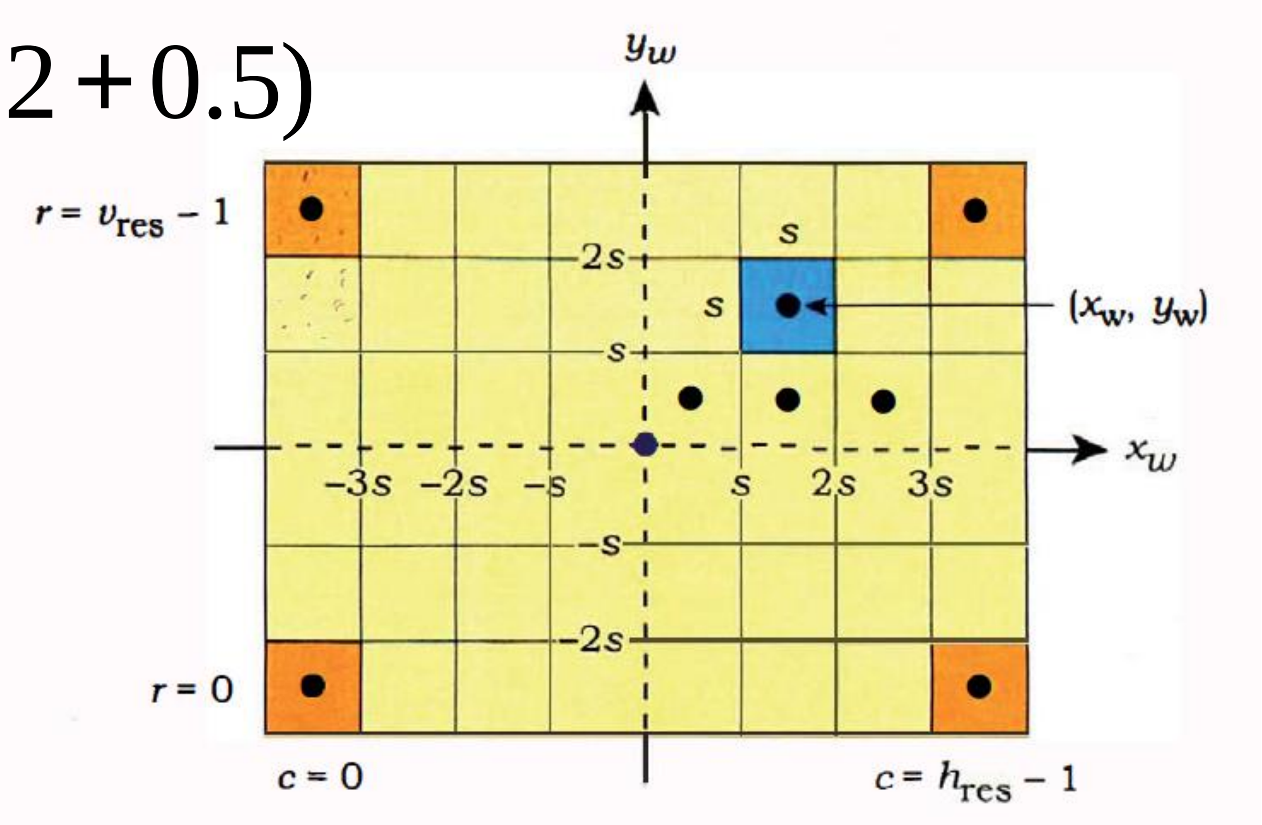
Coordinates are the signed area of the opposite subtriangle divided by area of the triangle



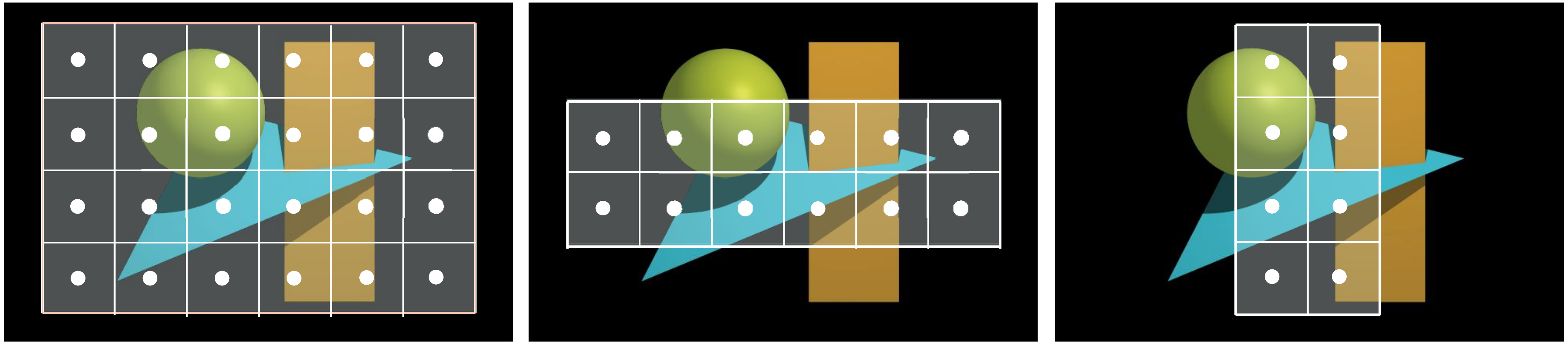
Computing Pixel Coordinates

$$x_w = s(c - h_{res}/2 + 0.5)$$

$$y_w = s(r - v_{res}/2 + 0.5)$$



Changing Field of View



Implementation Tips

- Don’t feel compelled to use the code from the book
  - It may be easier to write it yourself
- Choose a good, easy-to-use library for vector-matrix math
  - …or write your own
- Debug using single pixel images when possible
- Use test-driven development
  - As you incrementally add functions…
  - …write test executables for all major functions